Arduino 101
Hands-on: Tone

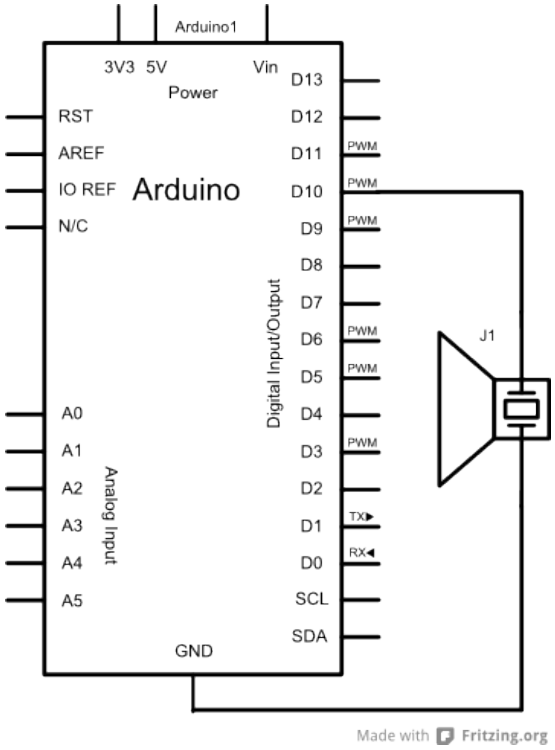
Project Description

This project will be a small diversion to demonstrate other ways the Arduino can be used for output. In this case, we're going to generate some tones on a piezo speaker.

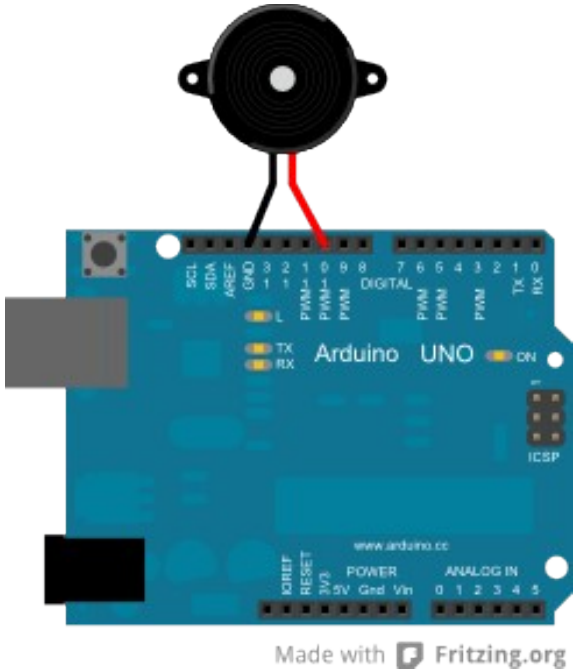
Required Parts

1 piezo element

Schematic



Circuit



NOTES: You don't need to remove the parts you have on your breadboard, but if you have a wire connected to the **GND** pin indicated on the diagram, then remove it before placing the piezo. Your piezo element should fit perfectly between the **GND** and **D10** header connections – just make sure you don't plug it into **AREF**!

Code

```
// Tone Example Sketch for Arduino 101
// by Nick Borko

void setup() {
  pinMode(10, OUTPUT);
}

void loop() {
  tone(10, 293, 250);
  delay(250);
  tone(10, 329, 300);
  delay(400);
  tone(10, 261, 350);
  delay(500);
  tone(10, 130, 450);
  delay(550);
  tone(10, 196, 1000);
  delay(5000);
}
```

Discussion

The **tone()** function can be used to generate tones on an output pin. It takes parameters for a pin number, frequency and, optionally, a duration. This is one way the Arduino can be used to generate audio output. There is also a **noTone()** function to stop any tones being generated.

The **tone()** function works by generating a square wave. This is done by rapidly changing the output pin from **LOW** to **HIGH** at specific intervals to generate the square wave at the specified frequency. The piezo element reacts by moving a metal element inside the housing when a **HIGH** signal is received, and this movement of the element generates the tones that you hear.

Our program is simple. We generate a tone for a specific interval, then we **delay()** to wait for that tone to finish playing, and sometimes a little extra time for a pause when no tone is playing. The frequencies for this musical phrase were found by using Google, so there wasn't any guesswork involved.

On a technical note, the Arduino Core uses the AVRs timer interrupts to precisely trigger the changes in output, so other operations that use timers, such as PWM, will not work correctly while the **tone()** is operating.

You can generate your own tones by using a loop to trigger the **LOW** and **HIGH** signals using **digialWrite()** and by using **delayMicroseconds()** in between to generate the proper frequency, but that is left as an exercise to the reader.